High Performance Embedded System Optimization Using Algebraic and Generalized Retiming Techniques

Miodrag Potkonjak, Sujit Dey
C&C Research Laboratories, NEC USA, Princeton, NJ

Zia Iqbal, Alice C. Parker
Dept. of EE-Systems, University of Southern California, Los Angeles, CA

Abstract: Retiming, algebraic and redundancy manipulation transformations are widely used in both the high level synthesis and the compilers fileds. We present a new approach on how these powerful transformations can be applied to improve the performance of embedded systems, by optimizing their latency and throughput.

A simple modification is sufficient to adapt both the Leiserson-Saxe retiming algorithm and the recently introduced ERB algorithm for the new task. We introduce a new negative retiming technique and the algorithm which coordinates this technique with both algebraic and redundancy manipulation techniques for latency optimization. The effectiveness of all discussed techniques is demonstrated on a set of "real-life" examples. Latency and throughput are improved by factors of 7.06 and 2.83 respectively, often with minimal or no additional hardware overhead.

1.0 INTRODUCTION

1.1 Embedded Systems

Embedded systems are computing systems which continuously interact with a surrounding environment. The environment mey be, for example an automotive system or aircraft, or a chemical processing system, or even another computing or communication system. High level and system level synthesis research have addressed synthesis of embedded systems several times during the last two decades [Bor88, Cho92, Sri92]. In particular, a number of successful contributions in design of interfaces between heterogeneous computational structures has been made [Bor88, Gup92]. However, only recently has embedded system design been recognized as one of the key synthesis tasks [Ayl92]. This is influenced by both an increased importance of the embedded system market for integrated circuits, consumer electronics, telecommunication and many others industries as well as recognition of the variety and complexity of the involved technical challenges.

Embedded systems often contain certain requirements which greatly complicate their design [Ayl92]. The successful development of an embedded system requires a framework in which both complex computing and environment modeling tasks can be appropriately addressed. The complexity of the computing tasks and need for cost effective solutions sometimes requires hardware-software codesign and partitioning. The environment modeling task imposes a need for both exact and approximate analytical methods as well as fast and accurate simulation techniques. Other important issues include appropriate embedded system debugging tools, and due to the portable nature of many embedded system optimization of power, size and weight.

By far, the two most restrictive and difficult requirements involve meeting the hard real time constraints and reactive nature of computation (execution of computation is in response to environment events which occur most often at continuous and cyclic pace). These two requirements make latency and throughput the key parameters for performance optimizationof embedded systems, which is the topic of this paper.

1.2 Modeling of Optimization of Latency for Embedded Systems

Figure 1 shows a typical embedded system where a computing system interacts with its environment. The embedded system consists of two parts: (i) controlling part and (ii) controlled part. The controlled part is a system which provides services or products to the end users and can be, for example, a mechanical, biological, or chemical system. System level or high level synthesis designers usually assume that all parameters of controlled part are specified and can not be altered.

The controlling part is a computation system which is needed to ensure proper functioning of the controlled system. We will concentrate on real-time embedded systems where the cost of implementation is only a secondary requirement and high performance is the ultimate design goal.

We will first define three parameters which are key for addressing the performance of real-time embedded systems: throughput, latency and critical path. The underlying assumed computation model is that a computation is done on a semi-infinite stream of incoming data. In the case of an embedded system, the input sample is produced by measurement done on the controlled part of the system. Furthermore, we will assume that there is only one input

1063-6404/93 $03.00 © 1993 IEEE

and one output, since generalization to support multiple inputs and multiple outputs is straightforward.

Definition 1: Latency (iteration period) is the number of time units needed to produce the corresponding output measured from the moment when the input sample is available.

Definition 2: Sampling period (initiation rate) is the minimum total number of time units between the acceptance of two consecutive input samples.

Definition 3: The critical path is the longest path between any input to the controlling part (either from the inputs of the controlled part or delays) to any output of the controlling part (either to its outputs or delays).

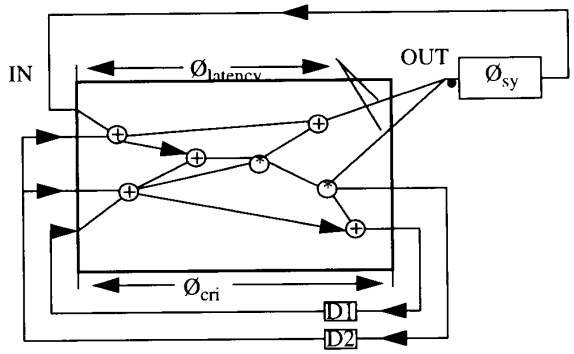

FIGURE 1. Embedded System: Controlling and Controlled part

Note that for non-embedded system the critical path is equal to the sampling period. We will denote latency by $\emptyset_{latency}$, sampling period by $\emptyset_{sample}$ and critical path by $\emptyset_{crit}$.

The embedded system operates in the following fashion. First an input sample is received by the controlling system from the controlled system. The controlling system executes the computation which is an implementation of the controlling algorithm. As soon as the output sample is computed a controlling signal is sent to controlled system which accordingly changes es its functioning. After the time $\emptyset_{sys}$, which is dictated solely by the nature of the controlled system, a new sample is obtained by the controlling system from the controlled system.

The quality of the embedded system is strongly correlated to the rate at which samples can be provided by controlling system. Note that throughput of embedded systems must satisfy the following two properties:

(1) $\emptyset_{sample} \geq \emptyset_{latency} + \emptyset_{sys}$

(2) $\emptyset_{sample} \geq \emptyset_{crit}$

Most often $\emptyset_{sys}$ is higher than $\emptyset_{crit}$, and the sampling rate and performance of embedded system is dictated by the latency. Note, that due to the fully closed loop between controlling and controlled part, the embedded system can not be pipelined at all.

1.3 Paper Organization

While almost all high level synthesis tasks can influence the throughput and latency of an embedded system, recent results in both high level synthesis and compiler research indicate that most often the influence of transformations is the most profound. We will concentrate mainly on the influence of retiming on latency and throughput. However, since it is apparent that various transformations interact, and often can amplify or subdue each other influence, we will aslo discuss the simultaneous application of several transformations.

We will show that the combined application of retiming with algebraic and redundancy manipulation is superior to isolated application of retiming. The same examplewill be used to illustrate the distinction between minimizing latency and maximizing throughput. Next, after presenting a simple modification which produces the optimal polynomial complexity algorithm for retiming for latency and adapts the ERB algorithm [Iqb93a, Dey92] for addressing latency optimization, we will concentrate on the main technical novelty of this paper: negative retiming and the use of negative retiming in the ERB framework. Before drawing conclusion and outlining several promising directions for future research, we will present experimental results and outline the used software platform.

1.4 Motivational Example

Figure 2a-e shows the second order direct form II IIR filter. Assuming that each operation takes one control cycle both latency and critical path are 4 control cycles. Our goal in the rest of this section will be to reduce latency as much as possible.

Using the modified Leiserson-Saxe algorithm we can obtain the structure shown in Figure 2b. Note that retiming substituted delay D1 by delays D3 and D4. Now the latency is only 2 cycles, which is an improvement by a factor of 2. In this situation, retiming can not be further used to reduce latency. However, if we first apply common subexpression replication of variable A (Figure 2c), followed by associativity on the addition on the direct path from input to the output, we will enable the application of retiming as shown on Figure 2e. The final structure has the latency of only 1 cycle. Note that while common subexpression replication and associativity did not improve latency (the latency of structures on Figures 2c and 2d is 2 cycles), they enabled the successive application of retiming which resulted in an additional improvement.

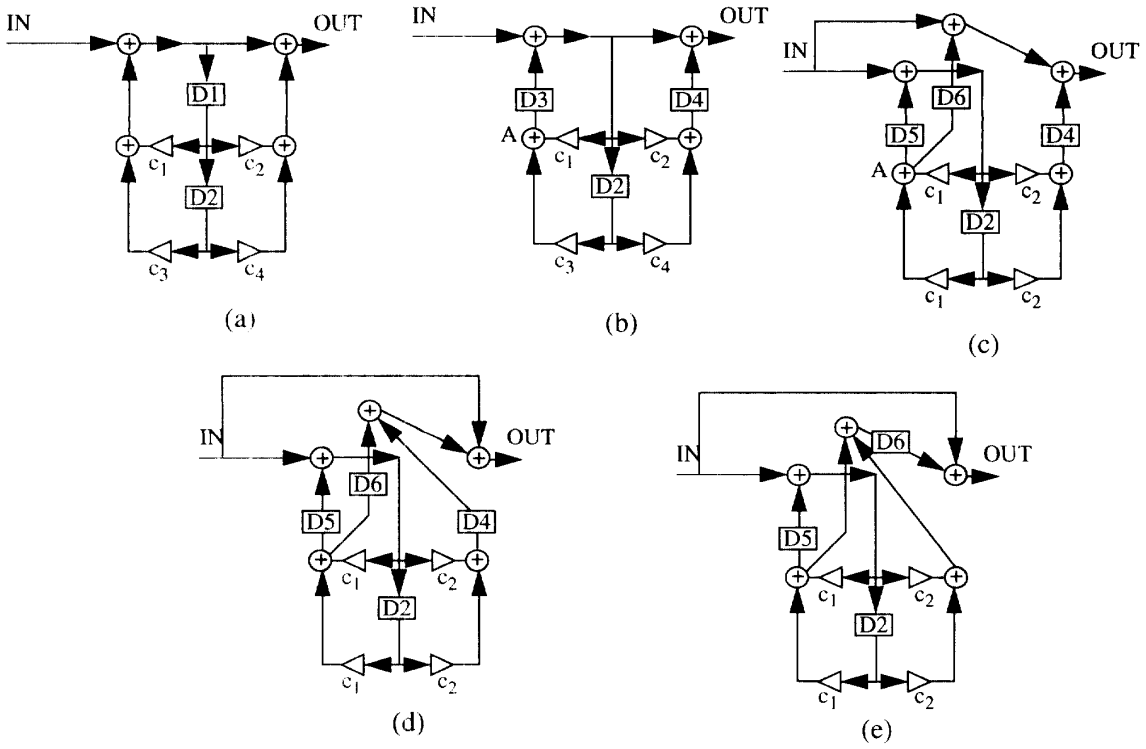

FIGURE 2. Retiming, algebraic and redundancy manipulation: (a) initial biquad; (b) after retiming; (c) after common subexpression replication; (d) after associativity; (e) after final retiming

All steps shown in Figure 2 can be automatically achieved by using first retiming for latency and then the ERB for latency algorithm as explained in the next section.

2.0 Retiming and ERB

Retiming [Lei83] is a transformation in which delays are added at some points in a computation and removed from others in such a way that the input/output relationship is not altered. Leiserson and Saxe [Lei83] presented several polynomial time algorithms which retime an arbitrary computation so that the critical path is minimized. Both the Leiserson-Saxe algorithm and several other retiming algorithms which address various other CAD optimization goals (e.g. resource utilization, power optimization) are successfully and widely used at several levels of the design process [Pot91, Cha92].

The ERB (elimination of retiming bottlenecks) is a critical path minimization technique which combines the power of retiming and algebraic and redundancy manipulation transformations, recently introduced in logic synthesis [Dey92], and modified, generalized and successfully applied in high level synthesis [Iqb93a, Iqb93b]. The ERB has three steps. In the first step sufficient timing condition to eliminate all retiming bottlenecks using algebraic and redundancy manipulation techniques are identified. These conditions are satisfied by the newly developed algebraic speed-up technique which uses algebraic and redundancy manipulations transformations. The final step is retiming so that a shorter critical path is achieved.

Modification of both retiming and ERB from optimizing critical path to minimizing latency or sample period is simple and straightforward. All what is necessary is to modify the initial computation so that all output nodes are connected to a new node, which has duration $\emptyset_{sys}$. Obviously if in this computation, critical path is shorter than some $\emptyset$, the sample period of the corresponding embedded system is also shorter than $\emptyset$. Vice verse is also valid, the sample period of the initial embedded system can be reduced to $\emptyset_{sample}$, only if the critical path of transformed computation can be reduced to $\emptyset_{sample}$.

An alternate way to adapt retiming and ERB for latency optimization, without modification of the considered computation, is to adapt the required and arrival times

of the output nodes in them, so that $\emptyset_{sys}$ is taken into account. This modification is again straightforward.

3.0 Enabling Algebraic Transformations using Negative Retiming

Algebraic transformations have been widely used in the past to optimize throughput of designs. However, their power is limited by the initial position of the delays. When algebraic transformations cannot improve performance any more, changing the position of the delays may "enable" the algebraic transformations again by exposing different parts of the flow graph for optimization.

The aim of the proposed technique is to perform retiming to *enable* algebraic transformations to further improve latency / throughput. The proposed process consists of an initial retiming, followed by algebraic transformations, followed again by a final retiming. The goal of the initial retiming is to move delays to the boundaries of the flow graph, so that the parts of the flow graph with high potential for algebraic optimization are exposed.

The conventional retiming technique by Leiserson and Saxe [Lei91], as we already mentioned, can be too restrictive, because it does not allow the use of negative delays. A delay on the fanin of a node will be not be able to move to the output of the node unless all the other fanins have a delay on them.

To allow more flexibility in retiming, the concept of peripheral retiming was introduced in [Mal91], and applied to logic circuits. Peripheral retiming allowed leaving negative number of latches (delays) at the inputs/outputs of the circuit, but it was shown that a subsequent retiming step can always remove the negative latches from the circuit, such that the final circuit is equivalent to the original circuit [Mal91]. Peripheral retiming can be also applied to flow graphs in the high level synthesis domain by substituting latches with delays and combinatorial with

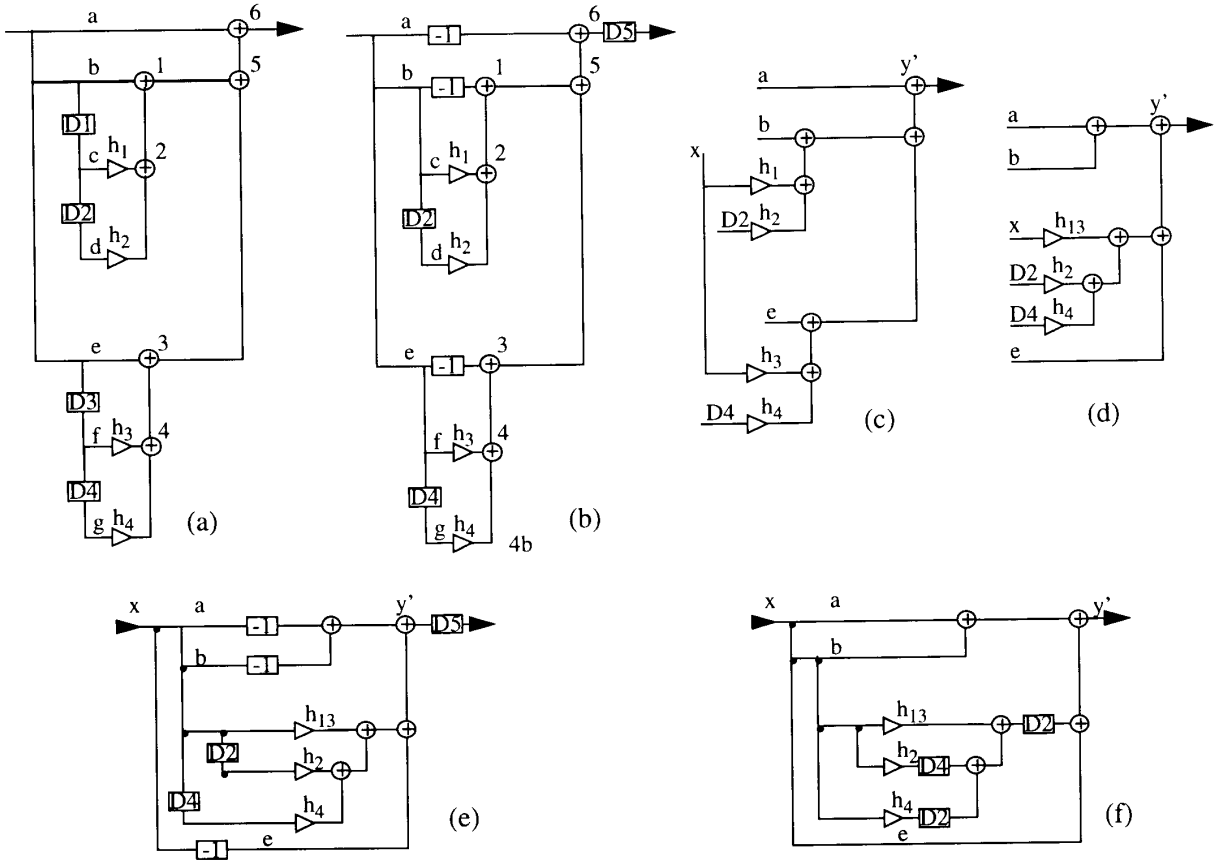

FIGURE 3. Enabling Algebraic Transformations and Retiming using Negative Retiming: (a) Original Flowgraph, (b) After Negative Retiming, (c) Extracted Sub-Graph, (d) Sub-Graph after Algebraic Transformations, (e) Transformed Flowgraph, (f) Final Flowgraph after Retiming

algebraic optimization. Alternatively, similar effect to one

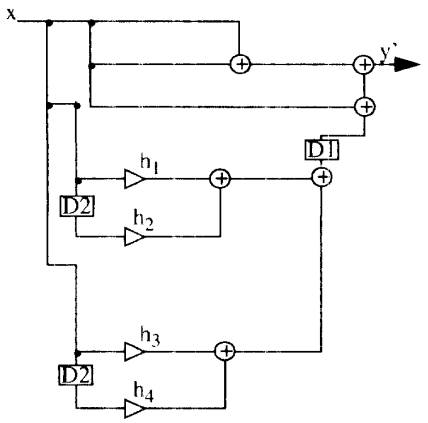

FIGURE 4. Flowgraph in Figure 3(a) after Algebraic Transformations and Retiming

produced by peripheral retiming, may be achieved by pipelining.

However, both pipelining and peripheral retiming may not always succeed to move all the delays to the outputs of the graph. Consider the flow graph shown in Figure 3a. Because of a lack of delay on the fanin *b* on node +*1*, the delay *D1* cannot be moved across node +*1* towards the output of the graph. Since the fanin *b* is an internal edge, neither peripheral retiming nor pipelining will help. Also, pipelining cannot be used to optimize embedded systems, since all the paths are in some loop; the primary inputs of the controlling system (the flowgraph to be optimized) come from the controlled system, and the primary outputs of the flowgraph go to the controlled system.

Hence, we introduce a more general retiming paradigm, where any edge, including internal edges, can have negative delays on them. The flow graph in Figure 3a illustrates the new retiming methodology. The delay D1 can be moved to the fanin of node +1, and after "borrowing" a delay from the internal edge *b*, can be moved to the output of +*1*. Similarly, a delay can be "borrowed" from the internal edge *e*, to allow delay D3 to move to the output of +*3*. Finally, borrowing a delay from the internal edge *a* would allow the delays to be moved to the primary output of the graph, as shown in Figure 3b. Notice that borrowing a delay from an edge corresponds to leaving a negative delay on the edge. A negative delay exists on the fanins a, b and e in Figure 3b.

As in the case of peripheral retiming, delays borrowed from the internal edges can always be returned later by performing the conventional Leiserson and Saxe retiming [Lei83]. It can be easily verified that the delay D5 at the primary output can be moved back, introducing a positive delay on the edges *a*, *b* and *e*. The negative and positive delays would cancel each other, thus clearing the flow graph of any negative delay.

3.1 The Optimization Process: An Example

The process of negative retiming, followed by algebraic transformations and subsequent retiming is illustrated using the data flow graph which is a part of IIR cascade and polyphase filters, shown in Figure 3.

Figure 3a shows the data flow graph, which is the controlling system in an embedded system. Let us assume that a multiplication with constant (denoted by a triangle) takes 2 control cycles, while an addition operation takes 1 control cycle. Since the longest path from either the Primary Input *x* or any of the delays to the Primary Output *y* is 6 control cycles long, the latency is 6. The length of the critical path, $\emptyset_{crit}$, is 6. Let us assume that the filter interacts with a controlled system, whose system time $\emptyset_{sys}$ is 1 control cycle. Since the sampling period $\emptyset_{sample} = \max\{(\emptyset_{latency} + \emptyset_{sys}), \emptyset_{crit}\}$, the sampling period for the embedded system is 7, being dominated by the latency of the controlling system.

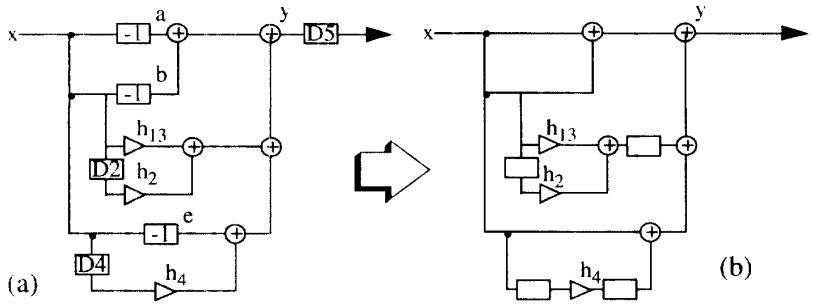

FIGURE 5. Minimizing critical path vs. Optimizing for Path Constraints: (a) Flowgraph after minimizing critical path, (b) Final Flowgraph after Retiming

Suppose that the desired sampling period of the embedded system is 3. Since $\emptyset_{sys}$ is 1, the final flow graph can have a latency no greater than 2, and the critical path length no greater than 3.

Retiming can be used to reduce the critical path length to 3. However, latency cannot be reduced to 2. In that case, the sampling period will be reduced to 4, being dominated by latency. Subsequent algebraic transformations does not help to reduce latency, thereby the sampling period.

On the other hand, algebraic restructuring, followed by retiming, can reduce the latency to 2. The resultant graph is shown in Figure 4. However, the critical path will be 4 control cycles long. Consequently, the sampling period of the embedded system will be 4 control cycles.

We now demonstrate how the process of negative retiming, algebraic transformations and conventional retiming can optimize the embedded system to achieve the desired sampling period of 3. The example shows how negative retiming can be used to enable algebraic transformations. Figure 3b shows the flow graph after negative retiming. The delays D1 and D3 have been moved to the output of the graph, leaving a negative delay on each of the edges *a*, *b* and *e*. Negative retiming has exposed a new region of the flow graph, shown in Figure 3c, which can be subjected to algebraic transformations. The subgraph is restructured to produce the subgraph shown in Figure 3d. Substituting the old subgraph by the transformed subgraph results in the flow graph in Figure 3e. Notice that the latency has been reduced to 2, while the critical path length remains to be 6. Subsequently, conventional retiming is applied to eliminate all the negative delays and minimize the critical path length.

The final flow graph, shown in Figure 3f, does not contain any negative delays, and has a latency of 2 and critical path length of 3 control cycles. Consequently, the embedded system has achieved the desired sampling period of 3.

3.2 Regions Exposed by Negative Retiming

The aim of negative retiming is to expose parts of the flow graph which can be optimized by algebraic transformations. It has been widely observed that reconvergent regions provide good opportunities for optimization. In the case of the flow graph in Figure 3a, the paths x-D1-h1-+2-+1-+5 and x-D3-h3-+4-+3-+5, starting at *x* and reconverging at node +5, form the reconvergent region. The paths of the reconvergent region have a delay each. Since algebraic transformations can only be applied within the boundaries of the delays, the reconvergent region cannot be transformed. However, if the delays D1 and D3 are moved by negative retiming to the primary output as shown in Figure 3b, the reconvergent region can be extracted as shown in Figure 3c. This allows the following transformation on the newly exposed logic:

$$x^* h_1 + x^* h_3 = x^* (h_1 + h_3) = x^* h_{13},$$

as shown in Figure 3d.

While various other criteria may be used to choose regions of the graph to be exposed (freed from delays), in this paper we aim to expose reconvergent regions by negative retiming.

3.3 Path Constraints for Algebraic Transformation

The aim of algebraic transformation applied to the exposed subgraph is not necessarily to minimize the length of the critical path. Figure 5a shows the flow graph which would have resulted if the critical path of the exposed subgraph of Figure 3c was minimized. A final retiming step would not be able to reduce the latency to 2, as shown in Figure 5b.

If the desired sampling period is $\emptyset_{sample}$, then the final flow graph after algebraic restructuring and retiming should satisfy the following path lengths. Any path from a PI or a delay to a PO should have length no greater than $(\emptyset_{sample} - \emptyset_{sys})$. Any path from a delay or PI to a delay should have length no greater than $\emptyset_{sample}$.

It is easy to see that a negative delay will always disappear after the final retiming step. The length of a path from a negative delay to the PO will be increased in the final graph by the path lengths to the negative delay. Consequently, while transforming the exposed subgraph, any path from a negative delay *X* to a PO should be reduced to $(\emptyset_{sample} - \emptyset_{sys})$ - a(X), where a(X) is the length of the longest path to *X*. We use the algebraic speed-up algorithm developed in [Iqb93a] to satisfy the path constraints.

3.4 The Optimization Algorithm

We present below the algorithm for optimizing embedded systems using negative retiming, algebraic transformations and conventional retiming. We term a reconvergent region to be of k-weight if each path of the reconvergent region has *k* delays.

k = 1;
repeat until (desired sampling period obtained) OR (all reconvergent regions exposed)
1. Expose k-weight reconvergent regions by negative retiming.
2. Perform algebraic speed-up to satisfy path constraints.
3. Perform conventional retiming to eliminate all negative delays, and optimize latency and critical path length.
4. k = k + 1;

4.0 Software Platform and Experimental Results

For the experimental verification of techniques and algorithms, we used the platform originally developed for critical path optimization using the ERB algorithm. This platform combines the strengths of the high level synthesis system Hyper [Rab91], the logic synthesis system SIS [Sen92], the originally developed logic synthesis program ERB [Dey92], and the high level synthesis program algebraic speed-up [Iqb93b]. This platform is described in detail elsewhere [Iqb93a].

Table 1 shows a set of 8 examples on which we tested the new techniques. The latency improvemed by anaverage factor of 1.37 for retiming, 3.56 for ERB and 7 06 for the technique which combines algebraic transformations and negative retiming (NERB). For a $\emptyset_{sys}$ of 3 cycles the sampling period improved by factors of 1.15, 2 10 and 2.83 for the retiming, ERB and NERB techniques respectively. The hardware overhead was very small for all techniques for all examples. The highest was for the NERB on the 4th order IIR where the number of additions increased by 12.5%, while the number of multiplications was unchanged. In several examples the NERB technique actually reduced the hardware requirements.

EXAMPLE	I	R	E	N
4 order IIR	9 / 6	7 / 4	5 / 2	4 / 1
adaptive IIR	9 / 6	9 / 6	5 / 2	4 / 1
Truncated Volterra	13 / 10	10 / 7	6 / 3	4 / 1
8th order Avenhaus	12 / 9	11 / 8	7 / 4	5 / 2
Cascade IIR	11 / 8	9 / 6	5 / 2	4 / 1
polyphase filter	8 / 5	8 / 5	5 / 2	4 / 1
speech filter	14 / 11	12 / 9	5 / 2	4 / 1
11th FIR	13 / 10	13 / 10	5 / 2	4 / 1

Table 1: Sampling period and latency for 8 examples: initial design (I), after retiming for latency (R), after ERB for latency (E), and after algebraic transformations and negative retiming (N). $\emptyset_{sys}$ is 3 cycles

5.0 Conclusion

We addressed the performance optimization of embedded systems using retiming, algebraic and redundancy manipulation transformations. The Leiserson- Saxe retiming algorithm and the ERB algorithm are modified to address the new latency optimization task. The new concept of negative retiming is introduced and a new algorithm which combines its power with algebraic and redundancy manipulation transformations is presented. On a number of examples this algorithm showed significant improvements.

6.0 References

[Ayl92] J. Aylor, R. Camposano, M. Schuette, W. Wolf, N. Woo: "The Future of Embedded System Designs", ICCD, pp. 144-146, October 1992.

[Bor88] G. Borriello, R.H. Katz: "Synthesis and Optimization of Interface Transducer Logic", ICCAD pp. 274-277, 1988.

[Cha92] A. P. Chandrakasan et. al: "Hyper-LP: A Design System for Power Minimization Using Architectural Transofrmations", ICCAD, pp. 300-303, 1992.

[Cho92] P. Chou, R. Ortega, G. Borriello: "Synthesis of the Hardware/Software Interface in Microcontroller-Based Systems", ICCAD pp. 488-495, 1992.

[Dey92] S.Dey, M. Potkonjak, S. Rothweiler: "Performance Optimization of Sequential Circuits by Eliminating Retiming Bottlenecks", ICCAD, pp. 504-509, 1992.

[Gup92] R.K. Gupta et. al. "Synthesis and Simulation of Digital Systems containing Interacting Hardware and Software Components", DAC, pp. 225-230.

[Hay89] S. Hayati, A. Parker: "Automatic Production of Controller Specifications From Control and Timing Behavioral Descriptions", DAC, pp. 75-80, 1989.

[Iqb93a] Z. Iqbal, et al.: "Critical Path Minimization Using Retiming and Algebraic Speed-Up", Technical Report #93-C003-4-5510-1, NEC USA, 1992.

[Iqb93b] Z. Iqbal, M. Potkonjak, S. Dey, A. Parker: "Critical Path Minimization Using Retiming and Algebraic Speed-Up", paper 33.2 DAC, 1993.

[Kai80] T. Kailath: "Linear Systems", Prentice Hall, Englewood Cliffs, NJ.

[Lei91] C.E. Leiserson, J.B. Saxe: "Retiming Synchronous Circuitry", Algorithmica, Vol. 6, pp. 5-35, 1991

[Lev91] N.G. Leveson: "Software Safety in Embedded Computer Systems", Communication of the ACM, Vol. 34, No. 2, pp. 34-46, 1991.

[Mal91] S. Malik, et. al."Retiming and Resynthesis: Optimizing Sequential Networks with Combinational Techniques", IEEE Trans. on CAD, Vol. 10, No. 1, pp. 74-82, 1991.

[Pot91] M. Potkonjak, J.Rabaey: "Optimizing for Resource Utilization Using Transformations", ICCAD, pp. 88-91, 1991.

[Rab91] J. Rabaey, et al. "Fast Prototyping of Data Path Intensive Architectures", IEEE Design and Test, Vol. 8, No. 2, pp. 40-51, 1991.

[Sen92] E.M. Sentovich, et. al: "Sequential Circuit Design using Synthesis and Optimization", ICCD, pp. 328-333, October, 1992.

[Sri92] M.B. Srivastava, T.J. Blumenau, R.W. Brodersen: "Design and Implementation of a Robot Control System Using a Unified Hardware-Software Rapid-prototyping Framework", ICCD, pp. 124-127, 1992.